NOVA IMS Information Management School

MDSAA

Master's Degree Program in

Data Science and Advanced Analytics

EasyML

An AutoML System using Meta Learning and Particle Swarm Optimization

Lyinder Nelson Swale

Dissertation

presented as partial requirement for obtaining the Master's Degree Program in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

EASYML

by

Lyinder Nelson Swale

Dissertation presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a Specialization in Data Science

Supervisor / Co Supervisor: Leonardo Vanneschi

February 2023

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, February 2023

DEDICATION

To my loving brother Rodgers, R.I.P.

ACKNOWLEDGEMENTS

I wish to thank God for his grace and strength while working on this research without which I wouldn't be able to complete successfully. I'm also very grateful for my supervisor LV who encouraged me and guided me through the right direction.

ABSTRACT

In recent years machine learning has made great strides in many application areas and an ever-growing number of disciplines rely on it. However, machine learning modelling process involves trying many machine learning algorithms with different parameter configurations which is considered insufficient, tedious, and time-consuming. The challenge has brought about the need for off-the-shelf solutions that allow a dataset to choose its best modelling pipeline including data preprocessing, model selection and hyperparameter optimization without or with very little human intervention in the process. Despite the availability of numerous AutoML systems that can automate the machine learning modeling process, there is still a need for a solution that can achieve the same results using a significantly smaller space, while improving efficiency. This thesis proposes an AutoML system named EasyML that uses meta-learning for model selection and particle swarm optimization for hyperparameter optimization. The research objectives include conducting a comprehensive literature review on State-of-the-Art techniques and existing AutoML systems, design, and development of EasyML, evaluating the system's performance on benchmark datasets, comparing its efficiency to other AutoML systems, and identifying its limitations and suggesting future research directions. The research methodology combines Design Science Research and CRISP-DM. EasyML outperforms existing solutions like SmartML and Auto-WEKA on all benchmark datasets. EasyML has the potential to contribute to the development of more efficient and effective AutoML systems, thereby meeting the increasing demand for data scientists with strong knowledge of various machine learning algorithms and techniques.

KEYWORDS

Automated Machine Learning; Meta Learning; Particle Swarm Optimization; Hyperparameter Optimization; OpenML

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AutoML	Automated Machine Learning
AI	Artificial Intelligence
PSO	Particle Swarm Optimization
ML	Machine learning
CASH	Combined Algorithm and Hyperparameter Search
SMAC	Sequential Model-based Algorithm Configuration
TPOT	Tree-based Pipeline Optimization Tool

1. INTRODUCTION

IBM Cloud Education[1], defined machine learning as the field of artificial intelligence and computer science that focuses on the use of data and algorithms to imitate the way humans learn, gradually improving its accuracy. The more data is available, the richer and more robust the insights and results that machine learning techniques can produce (Maher & Sakr, 2019).

In recent years machine learning has made great strides in many application areas (Feurer et al., 2015) and an ever-growing number of disciplines rely on it. In turn, there has been an increase in demand for data scientists with strong knowledge with various machine learning algorithms and techniques that can achieve the target performance and keep up with the exponential growing amounts of data produced daily. However, data scientists do not scale (Maher & Sakr, 2019).

Machine learning modeling process is a highly iterative exploratory process (Maher & Sakr, 2019) and there is no single model algorithm that performs best across all problems apriori. The process involves trying many machine learning algorithms with different parameter configurations which is considered insufficient, tedious, and time-consuming.

The demand for off-the shelf machine learning solutions that can easily be used without expert knowledge and support a more efficient and effective modelling process have brought about a relatively new field of research namely Automated Machine Learning (AutoML).

Several AutoML systems have been developed that allow a dataset to choose its best modelling pipeline including data preprocessing, model selection and hyperparameter optimization without or with very little human intervention in the process. SmartML (Maher & Sakr, 2019), Auto-WEKA (Hutter et al., 2018) and TPOT (Olson et al., 2016) are a few examples of existing AutoML frameworks available in the industry.

This research proposes a data driven approach to build an AutoML system with the purpose of improving efficiency. This was attained by limiting preprocessing, the use of meta learning for automatic algorithm selection to reduce the search space and particle swarm optimization for hyperparameter optimization to improve exploitation of the search space. The approach may save time and effort as it can automatically identify the best model and optimize its hyperparameters in an efficient and effective manner.

1.1. PROBLEM DEFINITION

Despite the availability of numerous AutoML systems that can automate the machine learning modeling process, there is still a need for a solution that can achieve the same results using a significantly smaller space, while improving efficiency. The problem is that the existing AutoML solutions have limitations in terms of the time and effort required to identify the best model and optimize its hyperparameters. Therefore,

[1] https://www.ibm.com/cloud/learn/machine-learning

the goal of this thesis is to design and develop a data-driven approach to build an AutoML system that uses meta-learning for automatic algorithm selection and particle swarm optimization for hyperparameter optimization, with the aim of improving the efficiency and effectiveness of the machine learning modeling process.

The thesis will investigate the scalability of the proposed system, its ability to handle diverse datasets and complex machine learning models, and its potential to improve the accuracy and performance of the models. The outcome of this thesis is expected to contribute to the development of more efficient and effective AutoML systems that can meet the increasing demand for data scientists with strong knowledge of various machine learning algorithms and techniques.

1.2. Research Objectives

To address the research goal, four main objectives were defined. Each objective seeks to support the problem at hand.

Objective 1: Conduct a comprehensive literature review on State-of-the-Art techniques and existing AutoML systems.

Objective 2: Design and develop a data-driven approach to build an AutoML system that uses meta learning for automatic model selection and particle swarm optimization for hyper parameter optimization.

Objective 3: Evaluate the performance of the proposed AutoML system using benchmark datasets and compare its accuracy with existing AutoML frameworks such as SmartML, Auto-WEKA, and TPOT.

Objective 4: Demonstrate the effectiveness and efficiency of the proposed AutoML system through experiments using real-world datasets.

Objective 5: Communicate the AutoML system's limitations and provide recommendations for future research and development in the field of AutoML.

1.3. Research Methodology

This research requires designing and developing an AutoML algorithm with the goal to improve the efficiency of AutoML process. The need for a more efficient way to perform predictive analytics has encouraged the development of this artifact. The algorithm will be researched to find out if it will prove its value by using a combination of two research methodologies namely Design Science Research and CRoss Industry Standard Process for Data Mining (CRISP-DM).

"*Design science…creates and evaluates IT artifacts intended to solve identified organizational problems*." (Hevner et al., 2004). It involves rigorous process to design artifacts to solve observed problems, to make research contributions, to evaluate the design and to communicate the results to appropriate audiences. On the other hand, the CRISP-DM framework is a task-focused approach used to execute data science projects. It aims to make large data mining projects less costly, more reliable,

more repeatable, more manageable, and faster. Figure 1.1 shows the crossover methodology framework between Design Science Research and CRISP-DM.

Problem understanding

This step includes problem identification, motivation, and objectives. The research problem which mainly is the need for an AutoML system that is more efficient, the motivation to design a solution that may potentially solve the problem and steps to take towards designing the solution were identified on the prior sections.

Design and development

In the design and development step, a scope literature review was conducted for the purpose of studying and analyzing related research work on meta learning and PSO in relation to AutoML. A proposed general meta learning architecture is used as the base for the design of the algorithm at hand.

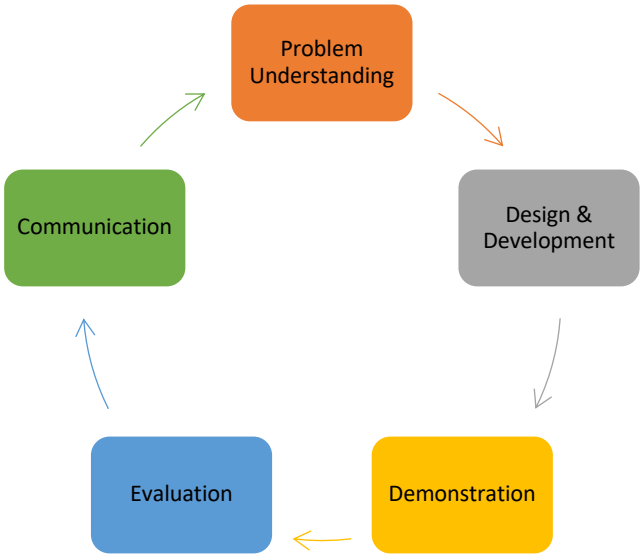

Figure 1.1 – Design Science Research and CRISP-DM Crossover

Demonstration

The algorithm was created using Python programming language and utilized various libraries such as OpenML for metadata sourcing, Scikit Learn for data pre-processing and machine learning models, and Glmnet and XGBoost for machine learning models. Auto-WEKA library was also used for some of the meta feature's creation. The input for the algorithm includes the dataset name and type, target feature, population size, dimension, minimum and maximum position, number of generations, and fitness criterion. The output includes the best predictive accuracy, the best hyperparameters, and the number of generations required to achieve the best accuracy.

Evaluation

The fully functioning algorithm underwent testing using four benchmark datasets to assess its performance. Upon comparison with some of the existing solutions, it was found that the algorithm demonstrated efficiency and effectiveness within its field of application. The algorithm performed well and was successful in achieving its intended goals when tested against standard datasets commonly used for benchmarking.

1.4. Achieved Results

The thesis achieved the goal of designing and developing an AutoML system, which can automatically identify the best model and optimize its hyperparameters in an efficient and effective manner, contributing to the development of AutoML systems that can meet the increasing demand for data scientists with strong knowledge of various machine learning algorithms and techniques. The system utilizes meta learning for model selection, leveraging prior knowledge obtained from over 600 diverse datasets and approximately 8 million machine learning tasks. To optimize the selected model, PSO is utilized. The system is constructed on top of seven classifiers and doesn't require preprocessing of the dataset.

1.5. Document Structure

The thesis is structured into seven chapters that provide a comprehensive study of the proposed AutoML system. The Introduction chapter provides an overview of the research problem, research objectives, and research significance. The State-of-the-Art chapter reviews the current state-of-the-art techniques and existing AutoML systems. The Methodology chapter explains the research methodology used to design and develop the proposed AutoML system. The Results and Discussion chapter presents the experimental results and compares the accuracy of the proposed system with existing AutoML frameworks. The Conclusion chapter summarizes the research findings and discusses their implications. The Limitations and Future Work chapter identifies the limitations of the proposed system and suggests areas for future research. The References chapter lists all references cited in the thesis.

2. STATE-OF-THE-ART

2.1. TRADITIONAL MACHINE LEARNING

ML involves the use of algorithms and statistical models to enable a computer to improve its performance on a specific task through experience, without explicit programming (Simon, 2015). Despite this claim, machine learning still requires considerable explicit programming (Olson et al., 2016). There are four main steps that go into building a ML model namely *data preprocessing*, *model selection*, *hyperparameter optimization* and *model evaluation*.

Data preprocessing is the process of preparing data for analysis by cleaning (Alasadi & Bhaya, 2017), transforming, and organizing it. It's an important step in the machine learning process because the quality and quantity of the data can affect the performance of the model significantly. This process includes data collection which involves gathering data from various sources such as databases or online platforms; data cleaning which involves identifying and correcting errors or inconsistencies in the data like missing values, duplicate entries, or incorrect data types; data transformation which involves data modification to make it suitable for analysis including data normalization or encoding; and organizing the data into a structured format that is easy to use and analyze.

Once the data is preprocessed, the next step is to select the appropriate model to train the data. Model selection is the process of choosing the best model form a set of candidate models for a given task. The main goal is to build a model that can make accurate predictions on new, previously unseen data. There are several techniques that can be used in model selection such as splitting data into training and test sets, using cross-validation and using a validation set to tune the hyperparameters for each model. This process is highly iterative as there isn't an ML model that performs best across all tasks and therefore the data must be tested on different models manually until the model with the best performance is found.

Once the model is selected, it's necessary to finetune the model's hyperparameters to achieve the best possible performance. Hyperparameter optimization is the process of finding the optimal values for the hyperparameters of a model. Hyperparameters are the parameters that are set before the model is trained, and they control the behaviour and performance of the model. Examples of hyperparameters include the learning rate, the number of layers in a neural network, or the regularization strength. Optimizing the hyperparameters of a model can significantly improve its performance, but it can be time-consuming and requires a lot of trial and error. There are several approaches to hyperparameter optimization, including manual tuning, grid search, and random search.

Model evaluation is an important step in the machine learning process, as it helps to identify areas of improvement and optimize the model for better performance. Model evaluation is the process of evaluating the performance of a machine learning model on a given task. This involves comparing the predicted outputs of the model with the actual outputs of the dataset and using various metrics and techniques to measure the accuracy and effectiveness of the model. Some common metrics used in model evaluation include accuracy, precision, recall, F1 score, and AUC (area under the curve).

Traditionally, these steps have required significant expertise in ML and data science. However, AutoML aims to automate as much of this process as possible, allowing even those with limited ML knowledge to build high-quality models.

2.2. AUTOMATED MACHINE LEARNING

Automated machine learning (AutoML) is a subfield of artificial intelligence (AI) and ML that focuses on automatically selecting and optimizing machine learning algorithms for a given dataset and task. It is an emerging field that aims to reduce human effort, improve accuracy and performance during the process of building machine learning models and promote the reproducibility and fairness of scientific studies (Hutter et al., 2018). AutoML benefits may be summarized as follows:

Automation of the machine learning process: AutoML allows users to automate the process of building, testing, and optimizing machine learning models, eliminating the need for expert knowledge in programming and data science.

Improved accuracy and performance: AutoML algorithms are designed to automatically optimize machine learning models for maximum accuracy and performance. This can lead to better results than manually built models.

Time and cost savings: AutoML can significantly reduce the time and cost required to build and optimize machine learning models, as it removes the need for manual coding and data preparation.

Increased efficiency: AutoML allows users to build, test, and optimize multiple quickly and easily machine learning models, increasing the efficiency of the machine learning process.

Greater accessibility: AutoML makes machine learning more accessible to non-experts, allowing individuals with limited programming and data science knowledge to leverage the power of machine learning in their work.

2.2.1. AutoML Techniques

AutoML may be applied through different approaches including:

i. *Automated data preprocessing* by automating the process of preparing data tasks such as missing values imputation, outlier detection and data scaling.
ii. *Automated model selection* by using algorithms to search through a space of possible models and select the one that performs the best on a given task.
iii. *Ensemble learning* which involves combining multiple ML models to improve the overall performance of the system.

iv. *Automated hyperparameter optimization* using algorithms to search through the space of possible hyperparameter values and select the ones that result in the best model performance.
v. *Neural architecture search* which involves using algorithms to search through the space of possible neural network architectures and select the one that performs the best on a given task.
vi. *Meta learning* which uses knowledge from prior tasks on new tasks in a data-driven manner; and
vii. *Full AutoML* which involves using algorithms to fully automate the entire ML model building process, from data preprocessing to model evaluation.

For the scope of this research, we will only discuss meta learning and hyperparameter optimization as well as related work done on the field.

2.3. META LEARNING

One key technique used in AutoML is meta learning, whose main goal is to develop algorithms that can adapt to new tasks and environments quickly using past experience (Hutter et al., 2018) and knowledge to inform the learning process. This allows the AutoML system to select the most appropriate model for a given problem based on its performance on similar tasks in the past.

In traditional machine learning, the algorithm is typically provided with a large dataset and left to learn the patterns and relationships within the data on its own. However, in meta learning, the algorithm is often provided with a smaller set of examples and must quickly adapt to a new task using the information from these examples.

If the AutoML system has previously seen a dataset with similar meta features to the current one, it can use the knowledge gained from that experience to select a model that is likely to perform well on the current task. This approach can save time and resources by avoiding the need to extensive trial-and-error in model selection.

Meta learning differs from traditional machine learning in the scope of the level of adaptation (Vilalta et al., 2009). Meta learning models are designed to learn from a small amount of data while traditional ML models typically require large amounts of data to be effective. Another key difference is the ability of meta learning models to learn from a wide variety of tasks and datasets whereas traditional ML models are specialized for a specific task at hand.

Efficiency is another factor distinguishes the two models where Meta learning models can adapt to new tasks more quickly and efficiently than traditional machine learning models, as they are able to transfer knowledge learned from previous tasks to new ones. Another difference is that meta learning models are used in situations where data or tasks change overtime while traditional ML models are used in static data.

2.3.1. Meta Learning Techniques

There are several approaches that can be taken into account while building a meta leaning system. (Vilalta et al., 2009) grouped the approaches into 6 categories namely *dataset characterization*, *mapping datasets to predictive models*, *learning from base-learners*, *inductive transfer and learning to learn*, and *dynamic-bias selection*.

Dataset characterization may be defined as a description of the properties of a dataset used to train an ML model for a particular task. Information acquired in this technique is relevant since high quality meta features provide some information to differentiate the performance of a set of given learning strategies (Vilalta et al., 2009). The characterization can include information about the distribution of the data, the types of inputs and outputs, the relationship between inputs and outputs, and the complexity of the task. Some of the data that can be extracted through dataset characterization include statistical and information-theoretic data such as number of classes, number of features, ratio of examples to features, degree of correlation between features and target concept, average class entropy and class-conditional en- tropy, skewness, kurtosis, and signal to noise ratio.

Landmarking and model-based characterization are other sources of data in dataset characterization. In landmarking, simple learners such as decision stumps or 1NN as used as models on a dataset and the accuracy and error rate of the models are used to characterize the dataset while in model-based characterization, a dataset is passed to a standard decision tree and the tree properties such as maximum tree depth, number of nodes per feature and tree imbalance are used to characterize the dataset.

A meta learning system has the ability of mapping a dataset to a predictive model using a predefined criteria such as accuracy, running time or storage space. In this technique, there are several approaches, but the most common ones are learning at the meta-level, mapping query examples to models and ranking. Learning at the meta level is when the learning system matches meta features of a new dataset to the existing ones usually contained in a *meta knowledge base* where each dataset has a target label associated with it and a predictive model is used to match the new dataset from the ones in the knowledge base by returning its target label.

In mapping query examples to models' approach, the meta learning systems uses the accuracy criteria to select models to exist in the knowledge base and once a new dataset enters the system, K-nearest neighbor approach is used where k best models for the datasets most similar to the new datasets are proposed. Ranking uses the same approach as learning at the meta level but returns more than one model for example it may return the first to fifth best performing models per dataset. This technique is thought as more flexible and informative for users.

Other techniques are learning from base learners where stacked generalization, boosting, landmarking and meta-decision trees approaches are used, inductive transfer where the new task is not matched to the existing datasets in the KB but rather the meta knowledge is incorporated in the new task and dynamic-bias selection which can be understood as the search for the right hypothesis space or concept representation as the learning system encounters new tasks.

2.3.2. Meta Learning Framework

(Vilalta et al., 2009) proposed a meta learning framework that is divided into two parts namely *Knowledge-acquisition mode* and *advisory mode*. In the Knowledge-acquisition mode, the system collects relevant data of various machine learning tasks from open-source platforms such as OpenML and Kaggle while advisory mode maps a new task metadata to the most similar existing tasks metadata using a similarity measure such as Euclidean distance[2] between the tasks.

OpenML is a platform for sharing machine learning datasets, code, and experiments (Vanschoren et al., 2014). It was developed to support researchers, developers, and data scientists in the machine learning community to easily collaborate and share their work. The platform allows users to discover and use machine learning datasets, algorithms, and experiments (Feurer et al., 2021) shared by other users, as well as to upload and share their own datasets, code, and experiments. It also provides tools for tracking and comparing the performance of different machine learning models on different datasets, and for finding the most effective models and techniques for a particular task.

Knowledge-acquisition mode

(Vanschoren, 2018) explores three approaches meta features can be used in the learning process i.e., model evaluations, task features and model configurations extracted from the tasks performed on different datasets. The main model evaluations meta features are pipeline and model configuration, predictive accuracy score, area under the curve and runtime. This process is useful for configuration recommendations and runtime predictions.

Tasks meta features may be used to find similarity between the existing tasks and the new task for the purpose of model or pipeline recommendations. This approach may also be used to find the relevance of a feature to a task's performance or even predict a model's performance. (Vanschoren, 2018) groups tasks meta features that can be extracted from a dataset into 6 categories namely simple, statistical, information-theoretic, complexity, model-based and landmarks. The categories are explained in detail on Table 2.1.

Model configuration-based features are mostly used for neural network training where techniques such as transfer learning and few-shot learning are applied. In this approach metadata comes from the machine learning models structures and parameters in such a way that a meta learner L that learns how to train a (base) learner l_{new} for a new task t_{new}, given similar tasks $t_j \in T$ and the corresponding optimized models $l_j \in L$ where L is the space of all possible models. The learner l_j is typically defined by its model parameters W = {wk}, k = 1...K and/or its configuration $\theta_i \in \theta$.

Table 2.1 – Task meta features and their measures.

Category	Feature	Measure

[2] Euclidean distance is the squared distance between points (Dokmanic et al., 2015)

Simple features	Dataset's dimensions (rows & columns) Target class distribution Missing values Numeric and categorical features Outliers	Speed & scalability Curse of dimensionality Complexity Data imputation
Statistical features	Data skewness Kurtosis Features correlation Covariance Sparsity PCA Class probability	Feature normality Degree of discreteness Feature dependence Class distribution
Information-theoretic features	Class entropy Normality entropy Mutual information Uncertainty coefficient Noise-signal ratio	Target class imbalance Feature informativeness Feature importance Noisiness of data
Complexity features	Fisher's discrimination Concept variation Data consistency	Class separability Task complexity Data quality
Model-based features	Number of nodes and leaves Branch length per class Information gain	Task complexity Feature importance Target class complexity Separability
Landmarkers features	1NN DecisionStump RandomTree NaiveBayes	Accuracy Error rate Data sparsity Separability Probing performance

Advisory mode

In advisory mode, knowledge base data acquired in the acquisition mode is matched with the meta features of the new dataset by a meta learner to produce a recommendation regarding the best available learning strategy. A meta learner may be defined as an algorithm that is trained how to learn from data for the purpose of selecting machine learning models for a given task.

The meta learner is trained on a large dataset of machine learning tasks, where each task is defined by a set of meta features as input data and a target output that can be a machine learning model, pipeline configuration, runtime etc.

A trained meta learner can be used to automate the process of selecting and training machine learning models for new tasks. To do this, the meta learner is given the input data and target output for the new task, and it uses its learned knowledge of how to select and train machine learning models to select the appropriate model and train it on the new data.

The goal of using a meta learner in AutoML is to automate the process of selecting and training machine learning models, which can be time-consuming and require a lot of domain expertise. By using a meta learner, the process of selecting and training machine learning models can be automated, allowing users to focus on other aspects of their machine learning project.

2.4. HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization (HPO) is a critical step in the development of machine learning models, as the selection of hyperparameters can greatly impact the performance of the model. There are several approaches to HPO, but the most popular ones are *grid search, random search, Bayesian optimization, gradient-based optimization, hyperband,* and *evolutionary optimization*.

Evolutionary optimization uses population-based methods, such as genetic algorithms, evolutionary algorithms, evolutionary strategies, and particle swarm optimization. These are optimization algorithms that maintain a population, i.e., a set of configurations, and improve this population by mutation and/or crossover to obtain a new generation of better configurations. These methods are conceptually simple, can handle different data types, and are extremely parallel since a population of N members can be evaluated in parallel on N machines (Hutter et al., 2018).

Particle swarm optimization (PSO) is a metaheuristic optimization algorithm inspired by the behavior of swarms in nature. The algorithm simulates animals' social behavior, including insects, herds, birds, and fishes (Wang et al., 2018). Each particle in the population represents different combinations of hyperparameter values of the machine learning model. The fitness of the particle is evaluated by training the model with the corresponding hyperparameters and evaluating its performance on a validation set. The personal best and global best solutions are updated as the particles explore the search space.

This approach allows the AutoML system to explore the vast space of possible hyperparameters quickly and efficiently and find the best settings for a given problem. This can improve the performance of ML models and reduce the need for manual tuning of the hyperparameters.

The main advantages of the PSO algorithm are summarized as: simple concept, easy implementation, robustness to control parameters and computational efficiency when compared with math mathematical algorithms and other heuristic optimization techniques (Lee & Park, 2006). This is because it is a population-based algorithm, which means it can explore the search space more effectively than a single-particle algorithm. This allows PSO to find the optimal solution in a more efficient and reliable way.

To use PSO for hyperparameter optimization, the search space is defined based on the range of values that the hyperparameters can take. The initial population of particles is then generated randomly

within this search space. The particles are then updated iteratively according to the PSO rules, which include a personal best and global best update rule.

At each iteration, each particle is updated based on its personal and global best positions and its current velocity. The personal best position is the best solution that the particle has found so far while the global best position is the best solution found by any particle in the population. The particle's velocity is updated according to a set of equations designed to steer the particle towards the global best position. This process continues until the algorithm converges to a satisfactory solution, or until a predetermined number of iterations is reached.

2.5. Related Work in AutoML

There are several AutoML systems in existence but for the scope of this study, three diverse systems that use different approaches are studied and analyzed.

2.5.1. SmartML

SmartML may be described as a meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms (Maher & Sakr, 2019) using meta learning technique for model selection and sequential model-based algorithm configuration (SMAC) for optimization. SmartML architecture (Maher & Sakr, 2019) consists of 5 phases namely input definition phase, data pre-processing phase, algorithm selection phase, parameter tuning and output computation and knowledge base updating.

In the input phase the user uploads a dataset, chooses the feature pre-processing techniques, and indicates the target feature of the dataset. The selected pre-processing techniques are then applied on the dataset, the data is randomly split in train and test data and meta feature computation is performed on the pre-processing phase. In the algorithm selection phase, the computed meta features are compared with the meta features in the knowledge base using a nearest neighbour approach then followed by measuring Euclidean distance between the new dataset and existing datasets followed by selecting n algorithms based on their performance. The selected algorithms are the optimized in the following phase where the original configurations from the selected algorithms are used as a starting point. The results from the optimization process are then compared and the best model configuration is returned to the user.

SmartML contains 15 machine learning classifiers and can be used as a package in R or as a web application.

2.5.2. Auto-WEKA

Auto-WEKA is an AutoML system that automatically and simultaneously chooses a learning algorithm and sets its hyperparameters to optimize performance using SMAC and tree structured parzen estimator (TPE). The tool was firstly created in 2013 for the purpose of solving the *combined algorithm selection and hyperparameter optimization* (CASH) problem (Kotthoff et al., 2017). Auto-WEKA

incorporates feature selection techniques and all machine learning approaches implemented in WEKA's standard distribution, spanning 2 ensemble methods, 10 meta-methods, 28 base learners, and hyperparameter settings for each learner.

The process begins with pre-processing the input data, which includes tasks such as handling missing values, normalizing the data, and feature selection. WEKA classifiers are then applied to the pre-processed data using different hyperparameter settings. These methods are then evaluated using a performance metric, such as accuracy or F1-score, and the best-performing combination of classifier and pre-processing method is selected for the final model.

Auto-WEKA can be run using the command line, GUI, or through a programmatic interface, and supports several performance metrics, search strategies and resampling methods for evaluating the performance of the models.

2.5.3. TPOT

(Olson et al., 2016) define Tree-based Pipeline Optimization Tool (TPOT) as a genetic programming based AutoML system that optimizes a series of feature pre-processors and machine learning models with the goal of maximizing classification accuracy on a supervised classification task. TPOT is a wrapper for the Python machine learning package, scikit-learn and uses 3 groups of operators namely feature selection operators, feature pre-processing operators and supervised classification operators.

Feature selection operators consists of VarianceThreshold, SelectKBest, SelectPercentile, SelectFwe, and Recursive Feature Elimination (RFE) while feature pre-processing operators are StandardScaler, RobustScaler, MinMaxScaler, MaxAbsScaler, RandomizedPCA, Binarizer, and PolynomialFeatures and supervised classification operators consist of DecisionTree, RandomForest, eXtreme Gradient Boosting Classifier, LogisticRegression, and KNearestNeighborClassifier. These operators are randomly combined using genetic programming and tested on a dataset and the combination or pipeline with the best performance is returned.

3. METHODOLOGY

The developed AutoML algorithm (EasyML) contains four main components namely the knowledge base (KB), the meta feature extractor, the meta learner, and the optimizer. For this study, the primary metadata is obtained through a designed pipeline that extracts real-time data from the OpenML repository. The collected data was used to create a knowledge base (KB) containing verified datasets basic data, complex data, tasks performed on the datasets their evaluations. The primary KB contained a total of 8 million tasks performed on 1216 datasets including their 96 meta features.

The (meta feature) extractor takes a new dataset as an input and returns tuple containing its meta features which are then passed to the meta learner which matches the meta features to the existing KB data and returns the best model of the most matching record. The return model then goes to the optimizer for optimization where the best hyperparameter values are returned. Python was used as the main programming language for building EasyML algorithm. Figure 2 is the architecture of EasyML system.

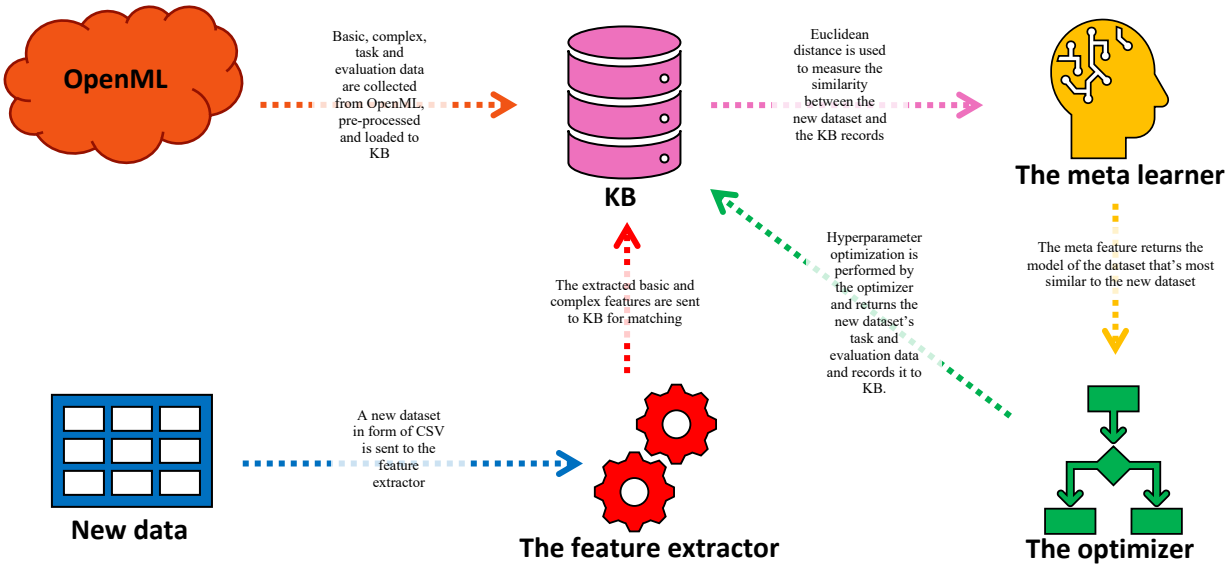

Figure 3.1 – EasyML Architecture

3.1. THE KNOWLEDGE BASE (KB)

In this section we describe the entire process of designing and building the KB. This includes data collection and the type of data collected, the detailed description of the data by exploratory data analysis and the techniques done to get the most relevant data required for the study such as feature selection, data filtering and aggregation by selecting records with the highest predictive accuracy values.

3.1.1. Data collection

The data used to create the KB was extracted through different queries which were integrated into one dataset. In this study, meta features were grouped into three categories namely basic data which contains simple features, complex data which is the combination of statistical, information theoretic, complexity, model based and landmarks meta features and lastly, task data which contains evaluations scores and pipelines configurations.

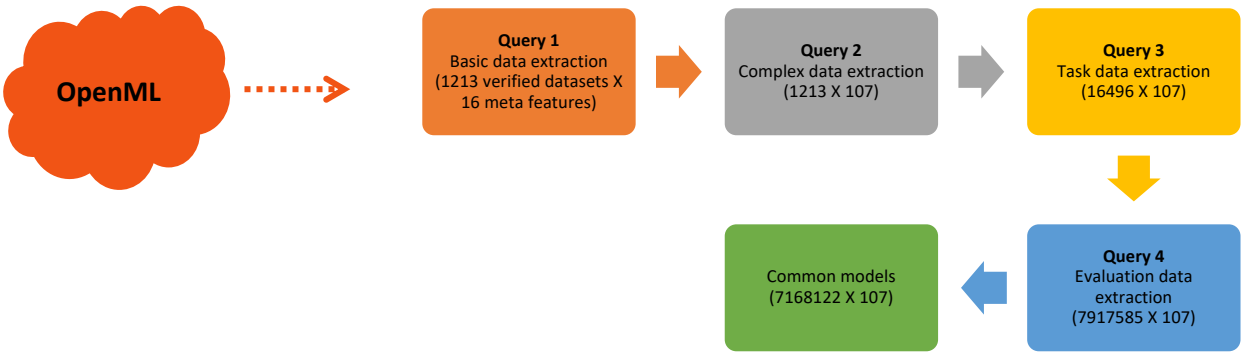

Figure 3.2 – Data collection process

Query 1 extracted 1213 verified datasets containing 16 basic meta features such as the datasets dimensions and the percentage of missing values.

```
# Extracting datasets basic data from OpenML Repository

openml_list = openml.datasets.list_datasets()  # returns a dict

dataset = pd.DataFrame.from_dict(openml_list, orient="index")

basic_df = dataset.dropna().reset_index().drop(columns ='index')
```

Figure 3.3 – Query 1 (basic data)

Query 2 followed by extracting the verified datasets complex data. In total 91 complex meta features were extracted which made the total of 107 features for each of the 1213 datasets.

```
# Extrating datasets complex data from OpenML Repository

complex_ = []
for i in basic_df.did:
    complex_.append(openml.datasets.get_dataset(i).qualities)

complex_df = pd.DataFrame(complex_)
```

Figure 3.4 – Query 2 (complex data)

Query 3 extracted all the tasks performed on the datasets and return the total of 16496 records being multiple tasks on each dataset.

```
# Extracting tasks performed on the datasetsfrom OpenML

tasks_df = openml.tasks.list_tasks(output_format="dataframe")
tasks = tasks_df[tasks_df['did'].isin(all_df['did'])]
tasks = tasks.sort_values('did')
tasks.rename(columns = {'did': 'data_id'},inplace = True)


# Grouping the tasks into lists chunks for the evaluations extraction process

tid = tasks.tid.to_list()
chunks = [tid[x:x+20] for x in range(0, len(tid), 20)]
```

Figure 3.5 – Query 3 (tasks data)

The tasks data was then used on query 4 to fetch close to 8 million experiments and runs performed on the tasks to return tasks evaluations in the form of predictive accuracy. The dataset was then filtered to keep only the most common models leaving the dataset with a little over 7 million rows of evaluation data. Figure 2 is an illustration of the process.

```
# Extracting tasks evaluations

evals = []
for i in range(1000):
    evals.append(evaluations.list_evaluations(tasks = chunks[i],
                                      function="predictive_accuracy",
                                      size=10000,
                                      output_format="dataframe"))

evaluations = pd.concat(evals)
```

Figure 3.6 – Query 4 (tasks evaluations)

Once all the data was extracted, ML models performed on the tasks were unified for a more accurate analysis. Random forest model for example appears as RandomForest, randomforest, randomForest or ranger. These discrepancies are relevant as Python language is case sensitive and would identify the items as different. Figure 8 is the program written for this task.

```
# Mapping models from the flows

random_forest = evaluations.loc[evaluations['flow_name'].str.contains('RandomForest') |
                                evaluations['flow_name'].str.contains('randomforest') |
                                evaluations['flow_name'].str.contains('randomForest') |
                                evaluations['flow_name'].str.contains('ranger')]
random_forest['model'] = 'RandomForest'


SVM_ = evaluations.loc[evaluations['flow_name'].str.contains('SVM') |
               evaluations['flow_name'].str.contains('svm') |
               evaluations['flow_name'].str.contains('SVC') |
               evaluations['flow_name'].str.contains('svc')]
SVM_['model'] = 'SVM'

XGBOOST_ = evaluations.loc[evaluations['flow_name'].str.contains('xgboost')]
XGBOOST_['model'] = 'XGBOOST'

GradientBoosting_ = evaluations.loc[evaluations['flow_name'].str.contains('GradientBoosting')]
GradientBoosting_['model'] = 'GradientBoosting'

DecisionTrees_ = evaluations.loc[evaluations['flow_name'].str.contains('DecisionTree') |
                                 evaluations['flow_name'].str.contains('rpart')]
DecisionTrees_['model'] = 'DecisionTree'

glmnet_ = evaluations.loc[evaluations['flow_name'].str.contains('glmnet')]
glmnet_['model'] = 'glmnet'

NaiveBayes_ = evaluations.loc[evaluations['flow_name'].str.contains('NaiveBayes')]
NaiveBayes_['model'] = 'NaiveBayes'

evals_models = pd.concat([glmnet_, DecisionTrees_, GradientBoosting_,
                          XGBOOST_, SVM_, random_forest, NaiveBayes_])
```

Figure 3.7 – Unifying tasks models

3.1.2. Data analysis and preprocessing

An in-depth analysis was performed on the data collected to understand data quality by checking the distribution of the target feature (ML model), missing values, outliers, and overall statistical data distribution.

Target feature distribution

The first analysis was performed to find the target class distribution by counting the frequency of each model across all tasks. Based on the data, the most used model is Random Forest followed by Support Vector Machine (SVM) while the least frequent model is Naïve Bayes followed by Gradient Boosting. Figure 4 is the graph showing the models distribution in the dataset.

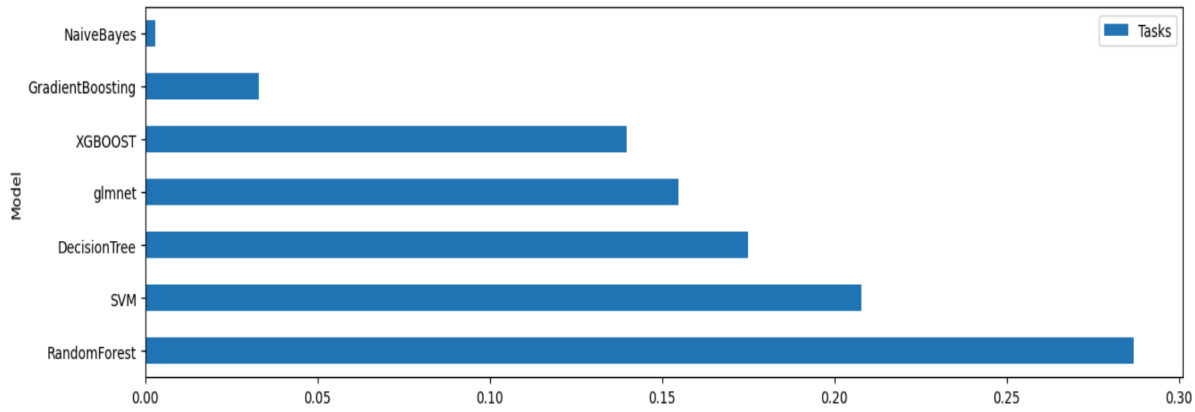

Figure 3.8 – Models distribution across tasks

Missing values

Another important analysis performed was on missing values. For a more effective approach, the analysis was performed on the dataset's columns. 80% of the columns contained less than 400 missing values while the remaining contained more than 600 missing values as shown on Figure 5. Based on this finding, all columns containing more than 400 missing values were removed.

```
# Removing columns with more than 400 missing values

dfy = complex_df.drop(
    complex_df.columns[
        complex_df.apply(
            lambda col: col.isnull().sum() >= 400)], axis=1)
```

Figure 3.9 – Removing missing values

Once the columns with >400 missing values were removed, a similar analysis was performed on rows (datasets) see Figure 6 and removed the rest of the missing values without a constraint. The remaining data contained the metadata of datasets without any missing values.

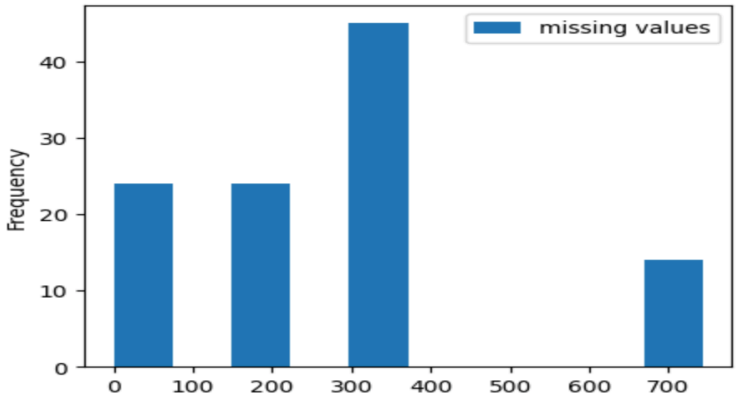

Figure 3.10 – Missing values by columns

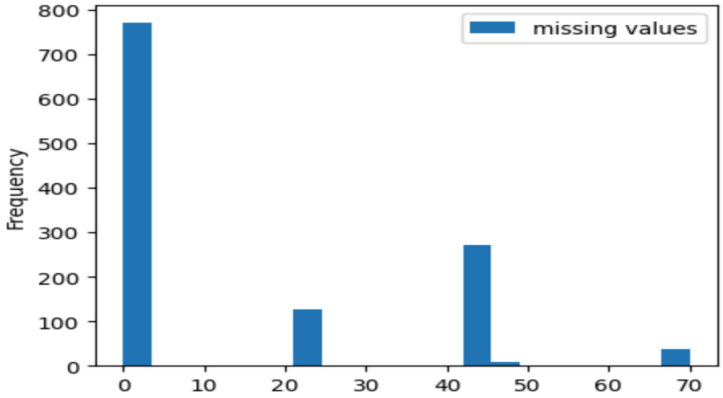

Figure 3.11 – Missing values by row

Selecting the best performing models

Tasks with the best predictive accuracy for each dataset were retained while removing the rest of the tasks records from the database. Here, it was discovered that some datasets had more than one best performing model. For tasks that had more than one best model, a simple ensembling algorithm was created to select the most common best model as final. Figure 7 shows the frequency for each best performing model.

A simple analysis was then performed to compare models' frequencies on all tasks versus best performing tasks and illustrated on Figure 8. A few observations were made from the comparisons:

- Random Forest model was used for most tasks and is also the model with most frequent best predictive accuracy.
- Although Naïve Bayes was the least used model, it is the second-most frequent best performing model across all tasks.
- SVM is consistent as it's frequency and performance rate are the same.

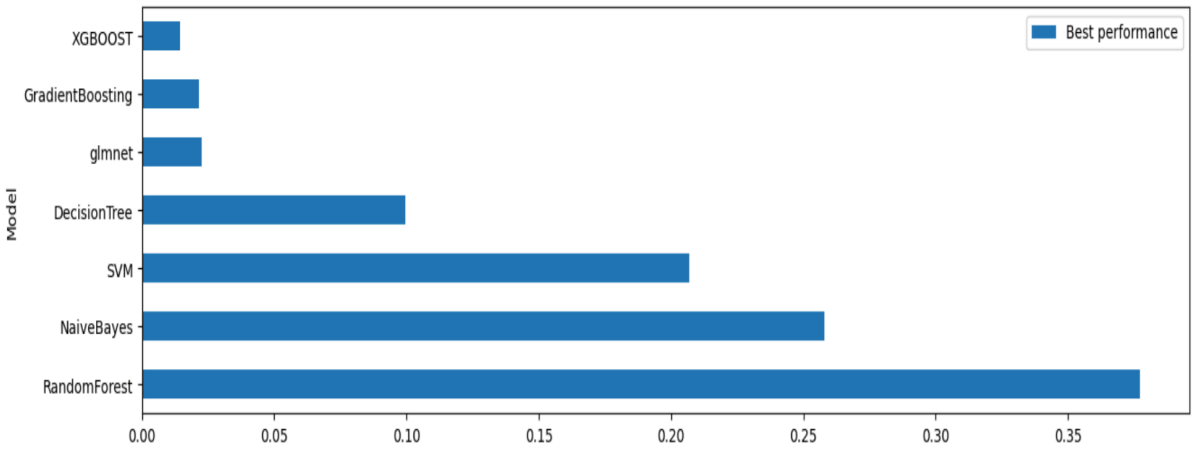

Figure 3.12 – Best models frequency across tasks

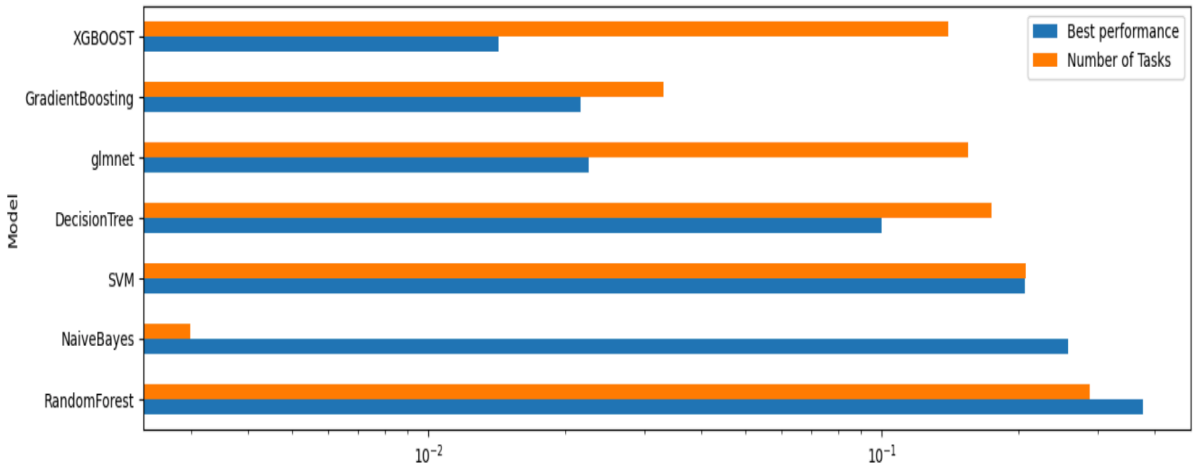

Figure 3.13 – Frequency of models' vs best performing models

The knowledge base consists of 680 datasets and 69 meta features for each dataset. Table 2 lists the meta features contained in KB with their respective categories.

Table 3.1 – Meta Features in KB

Category	Feature
Basic features	'NumberOfInstances', 'NumberOfFeatures', 'NumberOfClasses', 'NumberOfBinaryFeatures', 'PercentageOfBinaryFeatures', 'NumberOfSymbolicFeatures','NumberOfNumericFeatures', 'PercentageOfNumericFeatures','PercentageOfSymbolicFeatures', 'NumberOfMissingValues', 'NumberOfInstancesWithMissingValues', 'PercentageOfMissingValues', 'MaxNominalAttDistinctValues', 'PercentageOfInstancesWithMissingValues','MinorityClassPercentage', 'MajorityClassPercentage', 'MajorityClassSize', 'MinorityClassSize'
Complex features	'MaxKurtosisOfNumericAtts', 'MinKurtosisOfNumericAtts', 'MeanKurtosisOfNumericAtts', 'Quartile1KurtosisOfNumericAtts', 'Quartile2KurtosisOfNumericAtts', 'Quartile3KurtosisOfNumericAtts', 'MaxSkewnessOfNumericAtts', 'MinSkewnessOfNumericAtts', 'MeanSkewnessOfNumericAtts', 'Quartile3MeansOfNumericAtts' 'Quartile2SkewnessOfNumericAtts', 'Quartile2MeansOfNumericAtts', 'Quartile3SkewnessOfNumericAtts','Quartile1SkewnessOfNumericAtts', 'MaxStdDevOfNumericAtts', 'MinStdDevOfNumericAtts', 'MeanStdDevOfNumericAtts', 'Quartile1StdDevOfNumericAtts', 'Quartile2StdDevOfNumericAtts', 'Quartile3StdDevOfNumericAtts', 'Quartile1MeansOfNumericAtts', 'MaxMeansOfNumericAtts', 'MinMeansOfNumericAtts', 'MeanMeansOfNumericAtts', 'ClassEntropy', 'NaiveBayesErrRate', 'NaiveBayesKappa' 'J48_00001_ErrRate', 'J48_00001_Kappa', 'J48_0001_ErrRate', 'J48_0001_Kappa', 'J48_001_ErrRate', 'J48_001_Kappa', 'REPTreeDepth1ErrRate', 'REPTreeDepth1Kappa', 'kNN1NErrRate', 'kNN1NKappa','REPTreeDepth2ErrRate', 'REPTreeDepth2Kappa', 'REPTreeDepth3ErrRate', 'REPTreeDepth3Kappa', 'RandomTreeDepth1ErrRate', 'RandomTreeDepth1Kappa', 'RandomTreeDepth2ErrRate', 'RandomTreeDepth2Kappa', 'RandomTreeDepth3ErrRate', 'RandomTreeDepth3Kappa'

3.2. THE EXTRACTOR

A new dataset enters the system for the purpose of finding the model and hyperparameters that will give best predictive results based on its meta features although it doesn't have the features yet. The extractor is part of the AutoML algorithm that takes as input the new dataset and calculates 66 basic and complex features to match the existing meta features (see Table 2).

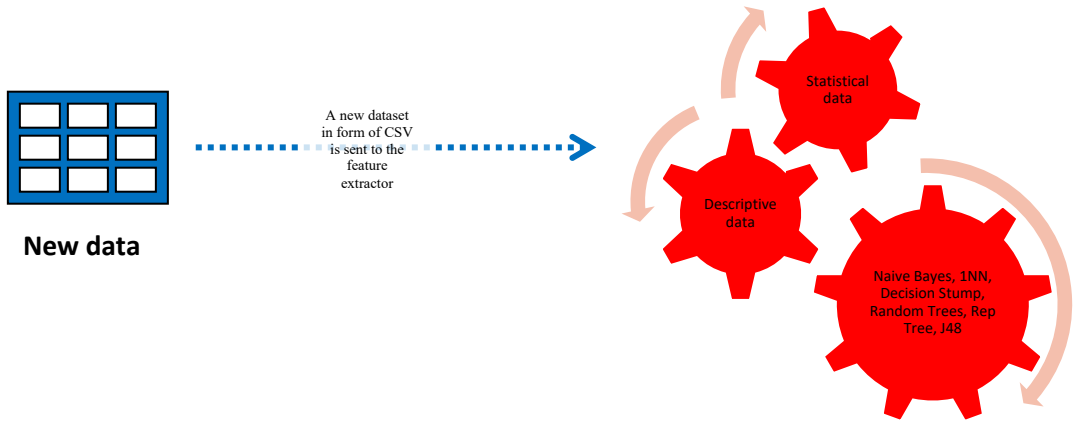

Figure 3.14 – The Extractor

Figure 3.15 is the one of the components of the main algorithm that extracts the landmarking meta features. Weka classifiers were mainly used as landmarkers by PyWeka library where the program would take a dataset, models and hyperparameters as an input and evaluations, error rate and Kappa scores were returned as datasets meta features.

```
# Calculates landmarking metadata

def landmarking(dataset, classifier, hypers):

    # Importing data
    loader = Loader(classname="weka.core.converters.CSVLoader")
    data = loader.load_file(dataset)

    # Setting target variable

    data.class_is_last()

    # Instatiating a classifier
    cls = Classifier(classname= classifier, options = hypers)

    # Evaluating data
    evl = Evaluation(data)

    # Cross Validating data with number of folds set to 10
    evl.crossvalidate_model(cls, data, 10, Random(1))
    results_summary = evl.summary()
    summary_df = pd.DataFrame(results_summary.replace('\n', ',').split(",")
                              [1:4])[0].str.split('       ',expand=True).fillna('')
    summary_df['value'] =  summary_df[1] + summary_df[2] + summary_df[3]
    summary_df['value'].loc[:1] = [summary_df['value'].loc[:1].str.strip().str.split(' ')[0][0],
                           summary_df['value'].loc[:1].str.strip().str.split(' ')[1][0]]

    summary_df['value'].loc[2:2] = summary_df['value'].loc[2:2].str.replace('     ', '')

    summary_df = summary_df.drop(columns = [1, 2, 3])
    summary_df['value'] = summary_df['value'].astype(float)
    errorRate = summary_df.value.loc[1] / (summary_df.value.loc[0] + summary_df.value.loc[1])
    Kappa = summary_df.value.loc[2]

    return evl, errorRate, Kappa
```

Figure 3.15 – Landmarkers component

3.3. THE META LEARNER

Given 680 tasks t_i, each with 66 meta features f_i^j and a target feature containing the task's best performing model β, the main objective of the meta learner is to select β that may perform best on the task t_{new} for the new dataset d_{new}. This is done by finding the task that is most similar to the new task by their meta features using the Euclidean distance E as the similarity function, the most similar task σ in the knowledge base K such that:

$$\forall i \in \mathrm{K} \qquad \mathrm{E}(\sigma, t_{new}) \leq \mathrm{E}(t_i, t_{new})$$

where:

$$\mathrm{K} = \{t_i,\ i = 1, 2, \ldots, 680\}$$

$$\mathrm{E}(S, t_{new}) = \sqrt{\sum_{i=1}^{n} (S, t_{new})^2}$$

β contains seven machine learning classifiers where 5 of them are from Scikit Learn library, one from XGBOOST and one GLMNET.

Table 3.2 – Classifiers search space

Classifier	Library
DecisionTreeClassifier	Scikit Learn
GLMNET	Glmnet Vignette
GradientBoostingClassifier	Scikit Learn
NaiveBayesClassifier	Scikit Learn
RandomForestClassifier	Scikit Learn
SVC	Scikit Learn
XGBoost	XGBoost

3.4. THE OPTIMIZER

The optimizer uses the global best ($\mathbf{G_{best}}$) PSO (Talukder, 2011) where the position of each particle is influenced by the best particle in the entire population. This approach leads to faster convergence as each particle connects with every other particle. The model selected by the meta learner (β) is passed to the optimizer where it's hyperparameters are tuned and modified using PSO to obtain the maximum accuracy score.

The model β represents a particle in the population which contains a vector of three continuous hyperparameters $\mathbf{X}$ that are dynamic and a vector of discrete hyperparameters ς that are static as they stay constant throughout the optimization process while $\mathbf{X}$ mutates. In this study, vector $\mathbf{X}$ represents the position of the particle β in the swarm. The optimizer consists of two main aspects namely the search space and the fitness function which will be discussed in the following section.

The optimizer initializes a population of n particles randomly where each particle has a current position in search space X_i, a current velocity V_i and a personal best position $P_{best,i}$ where i $\in$ [1, 2,...n]. The P_{best} corresponds to model β highest predictive accuracy when tested on the new dataset $\boldsymbol{d_{new}}$ using the fitness function $\boldsymbol{f}$. The position returning the highest accuracy among all $P_{best,\,i}$ becomes the $\mathbf{G_{best}}$.

The algorithm updates the positions of all particles depending on how the user sets the parameters. By default, EasyML creates 5 generations for the population of 5 particles. The number is set low to ensure the algorithm's efficiency. At the end of the algorithm, the position that returned the highest accuracy along with the accuracy are returned by the optimizer. The positions are updated by altering the particles' velocity using the following formula:

---

repeat for 5 generations

for each i = 1, 2, 3, 4, 5 do

$X_i := X_i + V_i$;

$V_i := w\,V_i + c_1r_1 * (P_{best,i} - X_i) + c_2r_2 * (\mathbf{G_{best}} - X_i)$;

if $(f(\beta(X_i)) > (f(\beta(P_{best,i}))$ then $P_{best,i} := X_i$;

if $(f(\beta(X_i)) > (f(\beta(\mathbf{G_{best}}))$ then $\mathbf{G_{best}} := X_i$;

end

end

The inertia weight w is picked randomly between the lower bound of 0.4 and higher bound of 0.9, the cognitive and social components (c_1 and c_2 respectively) are set constantly at 1 while r_1 and r_2 are picked randomly between 0 and 0.9.

This part of the algorithm performs two main tasks namely data preprocessing and predictive analytics on the new dataset $\boldsymbol{d_{new}}$. For efficiency purposes, data preprocessing techniques are limited to removing records with missing values, splitting the data to train and test datasets, and standardizing the data. random_state hyperparameter for the train_test_split method is one of the three

parameters that are dynamically altered by the algorithm. For data standardization, MinMaxScaler() and StandardScaler() are randomly selected. d_{new} is then trained by one of the selected 7 machine learning models and the test accuracy score is returned. For the tree-based algorithm, the other two hyperparameters represented as particle's position are max_depth and random_state while for SVC are cache_size and random_state.

4. RESULTS AND DISCUSSION

This chapter discusses the obtained results on the performance of our developed AutoML algorithm and its comparison to SmartML, Auto-WEKA and TPOT in terms of techniques and performance. The tests were performed with respect to efficiency by exploiting a small search space both by using seven ML models and optimizing as few hyperparameters as possible while returning better results than state-of-the art existing solutions.

The performance was tested on ten different configurations in order to understand it's behavior and what configurations would work better based on the dataset dimensions like the number of records, number of features and number of classes it contains. We tested the algorithm by setting the particles population and generations to 5 or 10 and maximum position to 10 or 20 or 50. The configurations were combined systematically, and the results were recorded and discussed below.

4.1. BENCHMARK DATA

We conducted experiments on four benchmark datasets to establish that the developed AutoML algorithm EasyML can find accurate models on a range of datasets in an efficient way. Results were collected on experiments performed on four datasets namely, Abalone, Madelon, Semeion and Yeast.

4.1.1. Abalone dataset

ABALONE is a well-known dataset of 9 continuous features and 4177 instances with no missing values where the goal is to predict abalones' sex whether its M, F, or I (infant). The dataset is modified data from research on population biology of abalones (1994), retrieved from UCI Machine Learning Repository.

The selected model for the Abalone dataset was Support Vector Machine where the optimized hyperparameters were cache_size and random_state. For pre-processing, the dataset was split 70/30 for train/test data and the random_state parameter was optimized. The best accuracy score was 59 when cache_size was 42, random_state for the model was 22 and random_state for the train_test_split was 26.

Table 4.1 – Abalone configurations and performance

Scaler	Population	Generations	Max pos	Best Pos	Max score
StandardScaler()	10	10	50	[42,22,26]	59
StandardScaler()	10	10	20	[6,13,15]	59
MinMaxScaler()	10	5	20	[1,11,15]	59
StandardScaler()	10	5	50	[11,35,42]	58
MinMaxScaler()	5	5	50	[39,7,34]	58

StandardScaler()	10	10	10	[6,9,8]	57
StandardScaler()	10	5	10	[4,2,8]	57
MinMaxScaler()	5	10	20	[41,18,27]	57
StandardScaler()	5	5	20	[7,4,18]	56
MinMaxScaler()	5	5	10	[2,4,10]	56

4.1.2. Madelon dataset

MADELON is an artificial dataset that is a two-class classification problem with 500 continuous input variables and 2600 instances. The dataset is a two-class classification problem, and its biggest challenge is that it's multivariate and highly non-linear. It was also retrieved from UCI Machine Learning Repository and was part of the NIPS 2003 feature selection challenge.

The selected model for the Madelon dataset was Random Forest where the optimized hyperparameters were max_depth and random_state. For pre-processing, the dataset was split 70/30 for train/test data and the random_state parameter was optimized. The best accuracy score was 75 when max_depth was 11, random_state for the model was 10 and random_state for the train_test_split was 22.

Table 4.2 – Madelon configurations and performance

Scaler	Population	Generations	Max pos	Best Pos	Max score
StandardScaler()	10	5	20	[11,10,22]	75
MinMaxScaler()	5	5	50	[42,33,23]	75
MinMaxScaler()	10	10	50	[16,25,3]	74
StandardScaler()	10	5	50	[34,8,3]	73
MinMaxScaler()	10	10	20	[10,23,23]	73
StandardScaler()	5	5	20	[18,12,9]	72
StandardScaler()	5	10	20	[12,11,15]	72
StandardScaler()	5	5	10	[9,1,2]	72
MinMaxScaler()	10	10	10	[9,7,6]	72
StandardScaler()	10	5	10	[4,5,9]	69

4.1.3. Semeion dataset

SEMEION is a multi-class classification task of a total of 1593 handwritten digits from around 80 persons were scanned, stretched in a rectangular box 16x16 in a gray scale of 256 values. Then each pixel of each image was scaled into a Boolean (1/0) value using a fixed threshold. The dataset contains 1593 records, 257 Boolean features and 10 target classes to predict.

The selected model for the Semeion dataset was Support Vector Machine where the optimized hyperparameters were cache_size and random_state. For pre-processing, the dataset was split 70/30 for train/test data and the random_state parameter was optimized. The best accuracy score was 97 when cache_size was 6, random_state for the model was 4 and random_state for the train_test_split was 3.

Table 4.3 – Semeion configurations and performance

Scaler	Population	Generations	Max pos	Best Pos	Max score
MinMaxScaler()	10	5	20	[6,4,3]	97
StandardScaler()	5	10	20	[10,2,9]	97
MinMaxScaler()	10	5	10	[5,3,8]	97
StandardScaler()	5	5	10	[5,10,9]	97
MinMaxScaler()	10	10	10	[10,1,9]	97
StandardScaler()	5	5	50	[46,39,25]	97
StandardScaler()	10	5	50	[21,24,9]	97
StandardScaler()	10	10	50	[42,19,9]	97
StandardScaler()	10	10	20	[4,7,9]	97
MinMaxScaler()	5	5	20	[5,1,17]	96

4.1.4. Yeast dataset

Yeast dataset consists of a protein-protein interaction network. The dataset has 1484 records, 8 features and 10 target classes which makes it a multivariate problem. It is to predict the cellular localization sites of proteins and was donated to UCI repository in 1996.

The selected model for the Yeast dataset was Random Forest where the optimized hyperparameters were max_depth and random_state. For pre-processing, the dataset was split 70/30 for train/test data and the random_state parameter was optimized. The best accuracy score was 68 when max_depth was 13, random_state for the model was 13 and random_state for the train_test_split was 13.

Table 4.4 – Yeast configurations and results

Scaler	Population	Generations	Max pos	Best Pos	Max score
MinMaxScaler()	10	5	20	[13,13,13]	68
MinMaxScaler()	5	10	20	[14,39,13]	67
MinMaxScaler()	5	5	20	[27,39,13]	67
StandardScaler()	10	5	10	[20,19,13]	66
StandardScaler()	5	5	50	[28,52,13]	66
StandardScaler()	10	5	50	[28,60,13]	65
StandardScaler()	10	10	50	[44,47,13]	65
MinMaxScaler()	10	10	20	[16,16,13]	65
StandardScaler()	10	10	10	[6,4,9]	64
StandardScaler()	5	5	10	[7,5,9]	62

4.2. COMPARISON OF EASYML TO OTHER AUTOML SYSTEMS

4.2.1. Performance

The results of the performance evaluation were compared to those of other AutoML systems, namely Auto-WEKA and SmartML, which had also been tested on the same benchmark datasets. For the Abalone dataset, EasyML performed significantly better where it was able to predict with 59% on the test data while Auto-Weka and SmartML had the score of 25% and 27% simultaneously.

For the Madelon dataset, EasyML performed slightly better than SmartML where the systems' scores were 75% and 74% respectively. Auto-WEKA gave the predictive score of 56%. EasyML performed better than both systems where it had the accuracy of 97% while SmartML performed at 94% and Auto-Weka at 89%. EasyML also performed best when tested on the Yeast dataset by predicting the target class by 68% while SmartML predicted at 66% and Auto-WEKA at 52%.

The table below summarizes the dimensions of the benchmark datasets used and performance on the three AutoML systems.

Table 4.5 - EasyML performance comparing to Auto-Weka and SmartML

Dataset	# Features	# Classes	# Instances	Auto-Weka Accuracy	SmartML Accuracy	EasyML Accuracy
abalone	9	3	8192	25	27	**59**
madelon	500	2	2600	56	74	**75**
semeion	256	10	1593	89	94	**97**
yeast	8	10	1484	52	66	**68**

4.2.2. Technologies

EasyML has several dependencies due to the technologies used. Besides Python and its poplar libraries such as Pandas and NumPy, the user must install Java virtual machine which supports the Weka library and install OpenML API to extract the most updated data from the repository. EasyML can be used locally and currently supports Python only.

In Table 4.6, a comparison of four different AutoML solutions, including EasyML, TPOT, Auto-Weka, and SmartML, is presented. The table displays information on the programming language used by each solution, the optimization method, the number of classifiers available on top of each language, and whether the solutions incorporate meta-learning or feature pre-processing. EasyML, which uses Python as its programming language, utilizes Particle Swarm Optimization for optimization and offers seven classifiers on top of Python.

On the other hand, SmartML, which is programmed in R, uses Bayesian optimization (SMAC & TPE) and offers 15 classifiers on top of R. Auto-Weka, programmed in Java, uses Bayesian optimization (SMAC) and offers 15 classifiers on top of WEKA. TPOT, which also uses Python, utilizes Genetic Programming for optimization and offers 27 classifiers on top of Sklearn. Both EasyML and SmartML incorporate meta-learning, while Auto-Weka and TPOT do not.

Table 4.6 EasyML technologies comparing to TPOT, Auto-Weka and SmartML

	EasyML	**SmartML**	**Auto-Weka**	**TPOT**
Programming language	Python	R	Java	Python
Optimization method	Particle Swarm Optimization	Bayesian (SMAC & TPE)	Bayesian (SMAC)	Genetic Programming
Number of classifiers	7 On top of Python	15 On top of R	27 On top of WEKA	15 On top of Sklearn
Use meta learning	Yes	Yes	No	No
Feature preprocessing	No	Yes	No	No

5. CONCLUSION

This research aimed to design and develop an AutoML algorithm that would improve the efficiency of the AutoML process. The research employed a combination of two research methodologies, Design Science Research and CRISP-DM, to achieve this objective. The methodology for developing the EasyML algorithm involved creating a Knowledge Base (KB) containing verified datasets, their basic and complex data, tasks performed on the datasets, and their evaluations. The data collection process involved four queries that extracted meta features from the OpenML repository, and a detailed data analysis was performed on the collected data to ensure its quality.

A fully functioning version of the algorithm, named EasyML, was implemented and evaluated using four benchmark datasets namely Abalone, Madelon, Semeion, and Yeast, and its performance was compared to SmartML, Auto-WEKA, and TPOT. The results show that EasyML outperformed these other AutoML systems on most of the datasets in terms of accuracy, and it did so with a small search space, optimizing only a few hyperparameters. The evaluation showed that the algorithm was efficient and effective in this field of application.

EasyML can help data scientists to perform predictive analytics more efficiently and the methodology employed in this research can be used as a framework for future research in the development of AutoML algorithms.

6. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

The EasyML algorithm is built on a knowledge base derived from primary metadata acquired via a designed pipeline that extracts real-time data from the OpenML repository, which serves as the backbone of the algorithm. However, the quality of data available in the repository has a significant impact on the algorithm's performance, given its heavy reliance on OpenML. In addition, the algorithm's limited data sources may constrain its ability to explore diverse datasets and tasks on other platforms such as Kaggle, potentially hindering its overall effectiveness.

It is important to note that the current implementation of EasyML has limitations that may impact its use in specific data science tasks. The algorithm is mainly designed for classification problems and lacks support for deep learning, thereby reducing its applicability in certain contexts. Moreover, EasyML cannot be used for clustering and regression problems, further limiting its use in certain cases.

To improve the performance and scope of the EasyML algorithm, various research and development opportunities exist. One such avenue is to expand the knowledge base by increasing the size and diversity of the datasets used in constructing it. This approach would enable the algorithm to leverage a wider range of experiences and patterns when making predictions and could involve integrating data from additional sources beyond the OpenML repository. Another direction for development is to extend EasyML capabilities to address other problem domains, such as regression or clustering, which would enhance its overall applicability.

7. REFERENCES

Alasadi, S., & Bhaya, W. (2017). Review of Data Preprocessing Techniques in Data Mining. *Journal of Engineering and Applied Sciences 12(16)*, *12*(16), 4102–4107.

Dokmanic, I., Parhizkar, R., Ranieri, J., & Vetterli, M. (2015). Euclidean Distance Matrices: Essential theory, algorithms, and applications. *IEEE Signal Processing Magazine*, *32*(6), 12–30. https://doi.org/10.1109/MSP.2015.2398954

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and Robust Automated Machine Learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 28). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf

Feurer, M., van Rijn, J. N., Gijsbers, P., Ravi, S., Müller, A., Vanschoren, J., & Hutter, F. (2021). OpenML-Python: an extensible Python API for OpenML. In *Journal of Machine Learning Research* (Vol. 22). https://docs.openml.org.

Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram, & Sudha. (2004). Design Science in Information Systems Research. *Management Information Systems Quarterly*, *28*, 75.

Hutter, F., Kotthoff, L., & Vanschoren, J. (2018). *Automatic Machine Learning: Methods, Systems, Challenges*.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. In *Journal of Machine Learning Research* (Vol. 18). http://automl.org/autoweka

Lee, K. Y., & Park, J.-B. (2006). Application of Particle Swarm Optimization to Economic Dispatch Problem: Advantages and Disadvantages. *2006 IEEE PES Power Systems Conference and Exposition*, 188–192.

Maher, M., & Sakr, S. (2019). SmartML: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. *Advances in Database Technology - EDBT*, *2019-March*, 554–557. https://doi.org/10.5441/002/edbt.2019.54

Olson, R. S., Edu, O., & Moore, J. H. (2016). *TPOT: A Tree-based Pipeline Optimization Toolfor Automating Machine Learning* (Vol. 64). https://github.com/rhiever/tpot.

Simon, P. (2015). *too BIG to IGNORE* (1st Edition). Wiley. https://support.sas.com/content/dam/SAS/support/en/books/too-big-to-ignore/69508_excerpt.pdf

Talukder, S. (2011). *Mathematical Modelling and Applications of Particle Swarm Optimization*. www.bth.se/com

Vanschoren, J. (2018). *Meta-Learning: A Survey*. http://arxiv.org/abs/1810.03548

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2014). *OpenML: networked science in machine learning*. https://doi.org/10.1145/2641190.2641198

Vilalta, R., Giraud-Carrier, C., & Brazdil, P. (2009). Meta-Learning - Concepts and Techniques. In *Data Mining and Knowledge Discovery Handbook* (pp. 717–731). Springer US. https://doi.org/10.1007/978-0-387-09823-4_36

Wang, D., Tan, D., & Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft Computing*, *22*(2), 387–408. https://doi.org/10.1007/s00500-016-2474-6

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa